Clase 2 — 27.10.25

🧠 Programación de Servicios y Procesos

👨‍🏫 Profesor: José Antonio Martín
Unidad: Programación Multiproceso
🗓️ Clase 2 — 27/10/2025
🎯 Tema: Procesos

0️⃣ PROCESOS

Un proceso es la **unidad de trabajo fundamental** del sistema operativo. Representa un programa que ha pasado de ser un simple archivo almacenado en disco a convertirse en una entidad activa. Para que esto ocurra, el sistema operativo debe asignarle recursos específicos, como memoria, tiempo de CPU, acceso a dispositivos y ficheros.

Cuando un programa se ejecuta, el sistema operativo crea una estructura interna que contiene:

- El código del programa.
- Los datos necesarios.
- Información del estado de ejecución.
- Recursos asignados.

Este conjunto permite que el proceso pueda iniciarse, pausarse, reanudarse y finalizar correctamente.

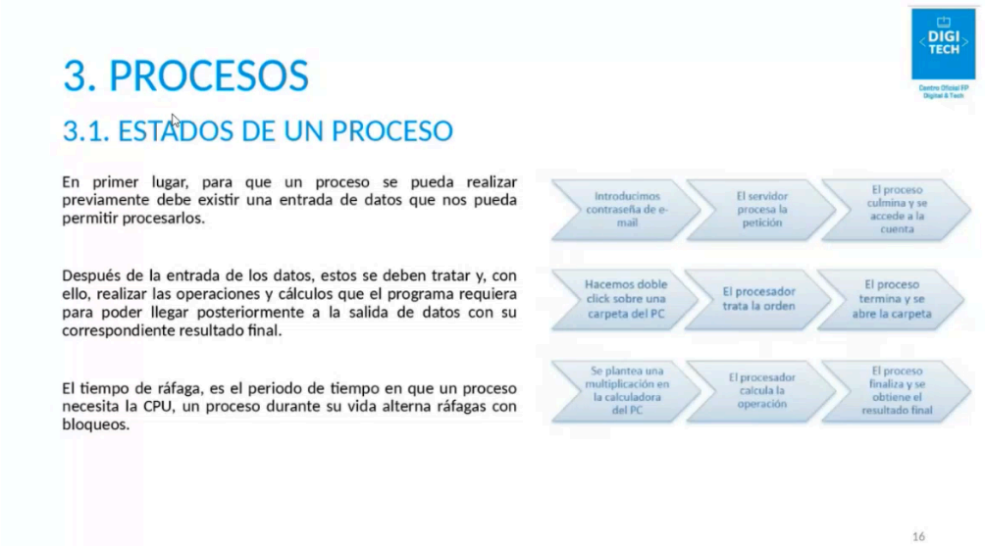
Cada proceso sigue un flujo definido:

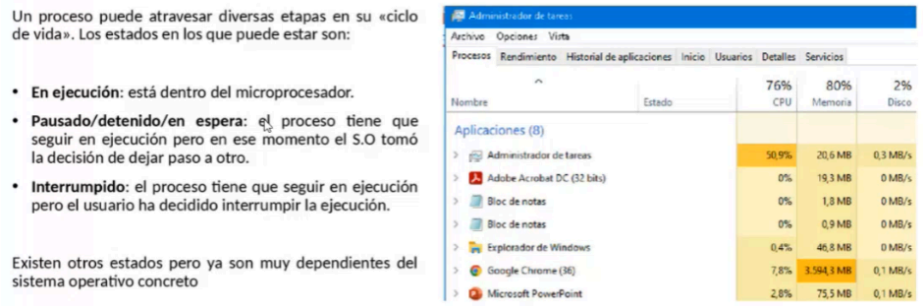
1. **Entrada de datos:** información proporcionada por el usuario, la red o el sistema.
2. **Tratamiento:** ejecución de instrucciones por la CPU.
3. **Salida:** resultado visible, almacenamiento o comunicación a otro proceso.

Los sistemas modernos pueden gestionar miles de procesos simultáneamente, muchos de ellos en segundo plano realizando tareas esenciales del sistema.

1️⃣ ESTADOS DE UN PROCESO

Los procesos cambian de estado continuamente según las decisiones del sistema operativo y las necesidades del programa.



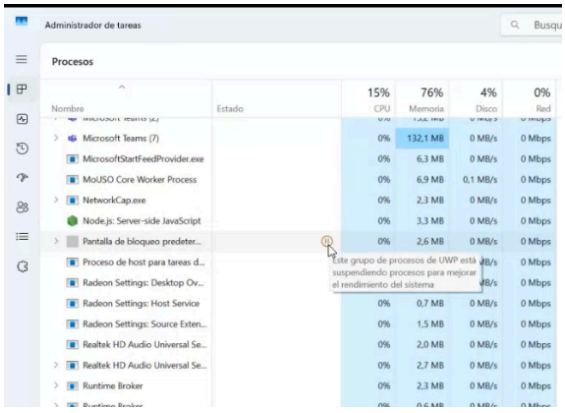


Parar un proceso

Parar significa que el proceso se detiene temporalmente, pero **su contexto completo se mantiene**. El sistema operativo guarda:

- El punto exacto donde quedó la ejecución.
- Los valores actuales de los registros.
- Su uso de memoria.

Cuando el sistema decide darle CPU de nuevo, regresa a ese punto como si nada hubiera pasado.



Interrumpir un proceso

Interrumpir implica detenerlo definitivamente. Es una acción explícita generada por el usuario o por el propio sistema. Al interrumpirlo:

- Su memoria es liberada.
- Su contexto se destruye.
- Sus recursos se devuelven al sistema.

Si el proceso se inicia otra vez, debe comenzar **desde el principio**.

Procesos visibles y en segundo plano

El usuario ve solo una parte del trabajo real del sistema operativo. Los navegadores modernos —como Chrome— dividen sus funciones en múltiples procesos para mejorar estabilidad y rendimiento. Esto significa que una sola aplicación puede generar decenas de procesos adicionales que trabajan en silencio.

--- DE UN PROCESO

Parar un proceso

El sistema operativo decide pausar el proceso. Mantiene su contexto y vuelve a ejecutarse desde el mismo punto.

Interrumpir un proceso

El proceso se detiene de forma definitiva. Si se ejecuta de nuevo, comienza desde cero.

Procesos visibles y en segundo plano

El Administrador de tareas muestra aplicaciones principales pero también **muchos procesos en background**, como servicios del sistema o procesos de navegadores como Chrome.

2️⃣ CICLO DEL PROCESO Y TIEMPO DE RÁFAGA

Para comprender cómo trabaja un proceso, es necesario analizar su comportamiento interno. Cada proceso alterna de manera constante entre fases de ejecución y espera, porque rara vez puede hacer todo su trabajo únicamente con la CPU. Muchas operaciones requieren acceder al disco, a la red, al teclado o a otros dispositivos, y esto genera pausas naturales.

Flujo general

Un proceso sigue un ciclo bien definido:

1. **Entrada:** obtiene datos, ya sea del usuario, de un archivo o de la red.
2. **Tratamiento:** la CPU ejecuta las instrucciones correspondientes.
3. **Operaciones y cálculos:** se procesan los datos, se ejecutan funciones y algoritmos.
4. **Salida:** entrega resultados al usuario o los almacena.
5. **Resultado final:** fin de la operación concreta.

Este flujo se repite constantemente mientras el proceso esté vivo. Un navegador, por ejemplo, está leyendo entradas (clics, teclas), realizando cálculos (interpretar HTML, ejecutar JavaScript), y generando salidas (dibujar en pantalla) de forma continua.

Tiempo de ráfaga (CPU Burst)

El tiempo de ráfaga representa el periodo en el que el proceso **tiene la CPU para él solo** y puede ejecutar instrucciones sin interrupciones. Ningún proceso está en CPU todo el tiempo. Alterna:

- **Ráfagas de CPU** donde calcula.
- **Periodos de espera (bloqueos)** donde necesita datos de E/S.

Importancia del tiempo de ráfaga

El sistema operativo utiliza este concepto para planificar. Los procesos con ráfagas cortas suelen ser interactivos (interfaces gráficas), mientras que los de ráfagas largas suelen ser computacionales

(cálculos intensos).

Ejemplos concretos:

- Abrir una carpeta: ráfaga de CPU breve, seguida de espera de disco.
- Renderizar un vídeo: ráfagas largas y sucesivas.
- Jugar a un videojuego: miles de ráfagas ultracortas sincronizadas con la GPU.

Comprender las ráfagas ayuda a diseñar políticas de planificación eficientes.

3️⃣ FUNCIONAMIENTO DE UN PROCESO

Un proceso no trabaja de forma aislada. Su ejecución implica interacción constante entre CPU, memoria y dispositivos.

Ejemplo 1: Abrir una carpeta

Cuando el usuario hace doble clic sobre una carpeta:

- La **entrada** es la acción del ratón.
- El sistema operativo **interpreta** esa acción y localiza la carpeta.
- La CPU calcula permisos, rutas y metadatos.
- La operación requiere acceso al **disco**, lo cual genera un periodo de espera.
- Tras obtener los datos, se muestra la carpeta.

Este ejemplo muestra un flujo que mezcla fases de cálculo y fases de bloqueo.

Ejemplo 2: Autenticación en un servicio

- Entrada: el usuario introduce una contraseña.
- Tratamiento: el programa prepara la petición.
- Cálculo: el servidor compara hashes.
- Lectura/escritura: accede a base de datos.
- Resultado: acceso concedido.

Cada parte implica subsistemas distintos del SO.

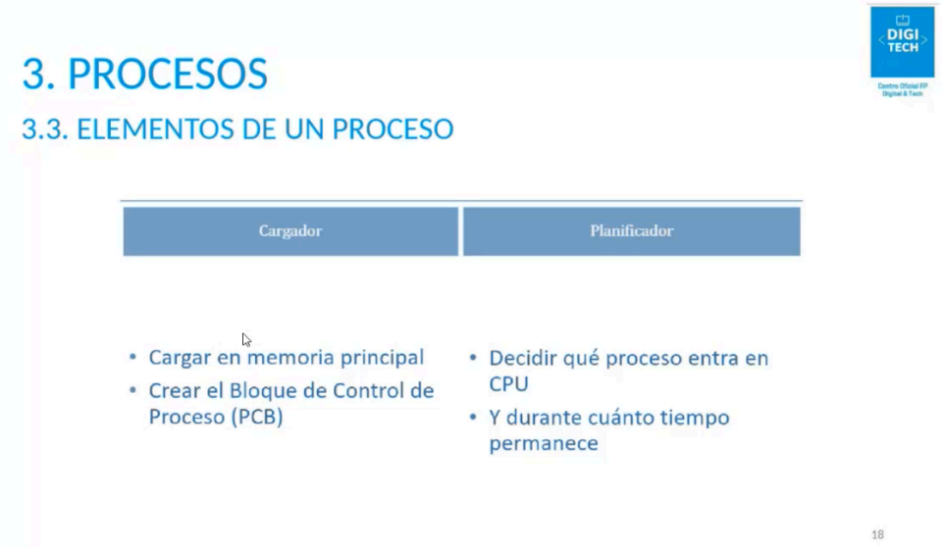
Ejemplo 3: Operación matemática

- Entrada: números.
- Cálculo puro: la CPU ejecuta la operación.
- Resultado: un valor en pantalla.

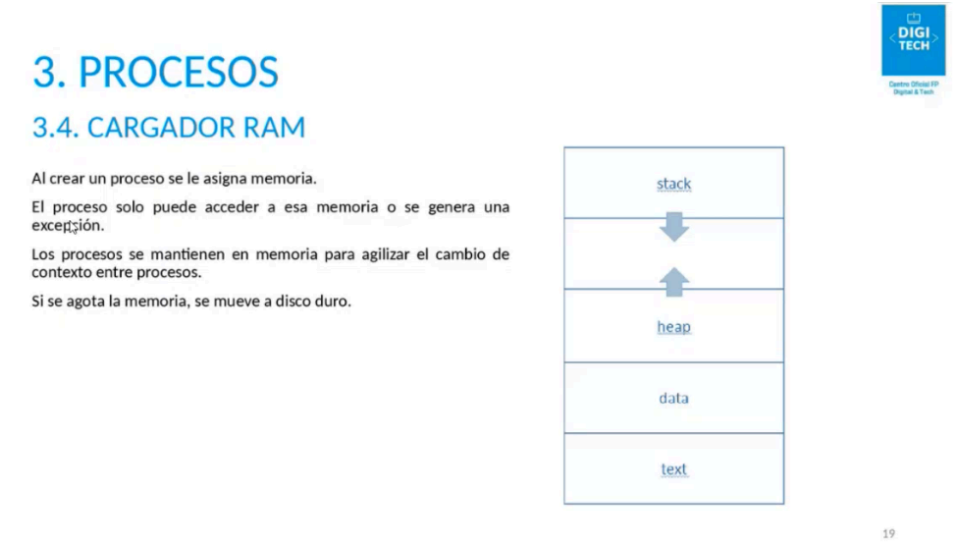
Este caso consiste casi exclusivamente en tiempo de ráfaga.

4️⃣ ELEMENTOS DE UN PROCESO

Los elementos que componen un proceso definen cómo se carga, cómo se ejecuta y cómo se planifica.



Cargador (Loader)



Es el responsable de transformar un programa almacenado en disco en un proceso listo para ejecutarse. Sus funciones incluyen:

- Reservar espacio en memoria para cada sección del programa.
- Inicializar las variables globales.
- Preparar el puntero de inicio (punto de entrada).
- Construir las estructuras de gestión como el PCB.

El cargador garantiza que todo esté correctamente organizado antes de que comience la ejecución.

Planificador (Scheduler)

Decide qué proceso debe usar la CPU en cada momento. Su trabajo es esencial para garantizar que el sistema sea:

- Eficiente.
- Justo.
- Reactivo.

Administra colas de procesos según prioridades o políticas (FIFO, Round Robin, SJF, etc.).

Quantum

Es el tiempo máximo que un proceso puede ocupar la CPU antes de que el sistema lo sustituya por otro.

- Un quantum pequeño mejora la respuesta, pero genera sobrecarga.
- Un quantum largo reduce cambios de contexto pero perjudica la interactividad.

El equilibrio depende del tipo de sistema.

--- DE UN PROCESO

Ejemplo

Abrir carpeta:

- Entrada: clic.
- Tratamiento: interpretación.
- E/S: acceso a disco.
- Cálculo: permisos.
- Resultado: carpeta visible.

Enviar contraseña en email:

- Entrada: usuario escribe.
- Tratamiento: envío a servidor.
- Cálculo: validación.
- Salida: acceso aprobado.

5️⃣ CARGADOR RAM Y CONTEXTO

La memoria es uno de los recursos más críticos para que un proceso pueda operar. Cuando un proceso se crea, el sistema operativo debe organizar su presencia en memoria de forma estructurada. Esta organización no es aleatoria: responde a un diseño pensado para maximizar rendimiento, seguridad y orden.

Regiones de memoria del proceso

La memoria de un proceso se divide en áreas con funciones muy específicas:

- **text:** contiene el código ejecutable. Es de solo lectura para evitar modificaciones accidentales.
- **data:** almacena variables globales inicializadas.
- **bss:** variables globales sin inicializar; se llenan con ceros al inicio.
- **heap:** espacio para asignación dinámica de memoria. Crece a medida que el programa solicita memoria (new, malloc).
- **stack:** contiene variables locales, direcciones de retorno y parámetros de funciones. Crece y decrece con cada llamada a función.

Esta estructura permite que el programa sea seguro y eficiente, evitando conflictos entre sus diferentes componentes.

Cambio de contexto

El cambio de contexto es uno de los mecanismos más importantes para la multitarea. Ocurre cuando el sistema operativo detiene un proceso para dar paso a otro.

Durante este proceso, el sistema debe:

1. Guardar el estado del proceso saliente.

2. Cargar el estado del proceso entrante.

El contexto incluye:

- Registros de la CPU.
- Contador de programa.
- Punteros de memoria del proceso.
- Información del planificador.

Cada cambio de contexto tiene un coste. Si sucede demasiado a menudo, reduce el rendimiento global del sistema.

Memoria virtual y movimiento al disco

Si la RAM se llena, el sistema recurre a la **memoria virtual**. Esto implica mover partes poco utilizadas de la memoria del proceso a una zona del disco llamada **swap**.

Ventajas:

- Permite ejecutar más procesos.
- Evita errores por falta de memoria.

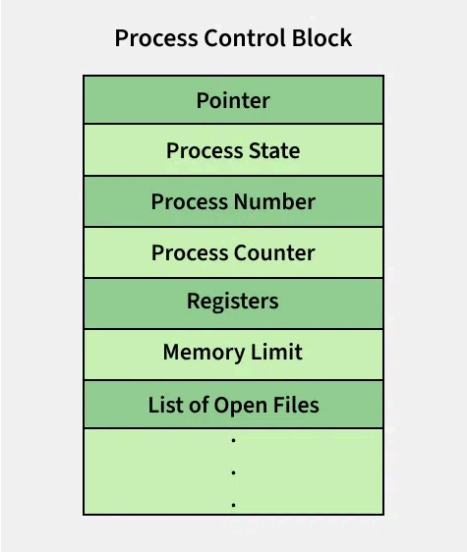
Desventajas:

- El acceso al disco es muchísimo más lento que a la RAM.

El sistema debe equilibrar qué regiones mantener en RAM y cuáles mover para que el usuario no perciba lentitud en exceso.

6️⃣ PCB Y MULTIPROCESOS

El **PCB (Process Control Block)** es la estructura central que utiliza el sistema operativo para administrar procesos. Funciona como el “expediente completo” del proceso. Cada vez que un proceso se pausa, se bloquea, cambia de estado o deja de usar la CPU, el PCB es lo que permite que pueda ser restaurado exactamente donde lo dejó.



A diferencia de otras estructuras del sistema, el PCB está pensado para contener **toda la información necesaria para reconstruir por completo el contexto de ejecución del proceso**, incluso en sistemas con miles de procesos concurrentes.

Componentes

✅ PID (Process Identifier)

Es un número único que identifica al proceso. Permite que el sistema operativo lo distinga entre cientos de procesos. También sirve para operaciones como enviar señales, rastrear jerarquías (PPID) o gestionar permisos.

✅ Estado

Indica la fase del ciclo de vida del proceso (nuevo, preparado, ejecutando, bloqueado, terminado). El planificador utiliza esta información para decidir qué proceso debe pasar a la CPU.

✅ Program Counter (PC)

Es la **pieza más crítica del PCB**: almacena la dirección exacta de la próxima instrucción que debe ejecutar el proceso. Si un proceso se pausa, al reanudarlo **se continúa desde esta dirección**, no desde el inicio.

✅ Registros de CPU

Incluyen:

- Registros de propósito general.
- Registros especiales (SP, BP, flags…).
- Registros de segmento o tablas de paginación según la arquitectura.

Estos registros deben guardarse y restaurarse para que el proceso continúe con valores correctos. Si no se recogieran, el proceso se corrompería automáticamente.

✅ Prioridad

Define la preferencia del proceso respecto a otros. Los planificadores basados en prioridades consultan este valor constantemente.

✅ Memoria asignada

El PCB indica dónde están ubicados los segmentos del proceso:

- **text:** contiene el código ejecutable del programa. Es de solo lectura para evitar modificaciones.
- **data:** almacena variables globales inicializadas con un valor definido antes de ejecutar el programa.
- **bss:** reserva espacio para variables globales no inicializadas. El sistema las rellena con ceros al arrancar el proceso.
- **heap:** región destinada a la memoria dinámica solicitada durante la ejecución (malloc, new). Crece hacia arriba según las necesidades del programa.
- **stack:** área usada para funciones, parámetros y variables locales. Crece y decrece con cada llamada o retorno de función.

También incluye referencias a:

- Tablas de páginas.
- Segmentos asignados.
- Información sobre memoria virtual.

Sin esto, el SO no podría mapear correctamente el proceso en la RAM.

✅ Ficheros abiertos

Cada proceso mantiene una tabla interna sobre los archivos o dispositivos que tiene abiertos:

- **sockets:** conexiones de red abiertas que permiten comunicación entre procesos o con servicios externos.
- **archivos:** ficheros en disco que el proceso está leyendo o escribiendo mediante descriptores.
- **pipes:** canales de comunicación unidireccionales usados para enviar datos entre procesos relacionados.
- **dispositivos especiales:** interfaces como `/dev/null`, terminales, puertos o dispositivos virtuales gestionados por el kernel.

Esto permite que el proceso mantenga control sobre todos sus recursos abiertos:

- Continuar operaciones E/S sin perder progreso.
- Cerrar recursos si el proceso muere.

¿Por qué el PCB permite pausar y reanudar procesos?

Cuando un proceso es interrumpido, toda su información vital se guarda en el PCB:

- **Estado:** indica en qué fase se encontraba el proceso (preparado, ejecutando, bloqueado) justo antes de ser detenido.
- **Registros:** el valor exacto de los registros de la CPU, necesarios para que el proceso continúe con cálculos pendientes sin errores.
- **PC (Program Counter):** la dirección de la próxima instrucción que debía ejecutarse; permite retomar la ejecución sin reiniciar el proceso.
- **Contexto de memoria:** información sobre segmentos, tablas de páginas y posiciones en memoria para restaurar el entorno de ejecución del proceso.
- **Ficheros:** todos los recursos de E/S abiertos, permitiendo continuar operaciones pendientes al reanudarse.

Después, el planificador puede asignar CPU a otro proceso. Más tarde, cuando el proceso original vuelva a ejecutarse, el sistema lee su PCB y **restaura su contexto al milímetro**, permitiendo continuar la ejecución como si nunca hubiese sido detenido.

Este mecanismo hace posible la multitarea real.

Multiprocesos

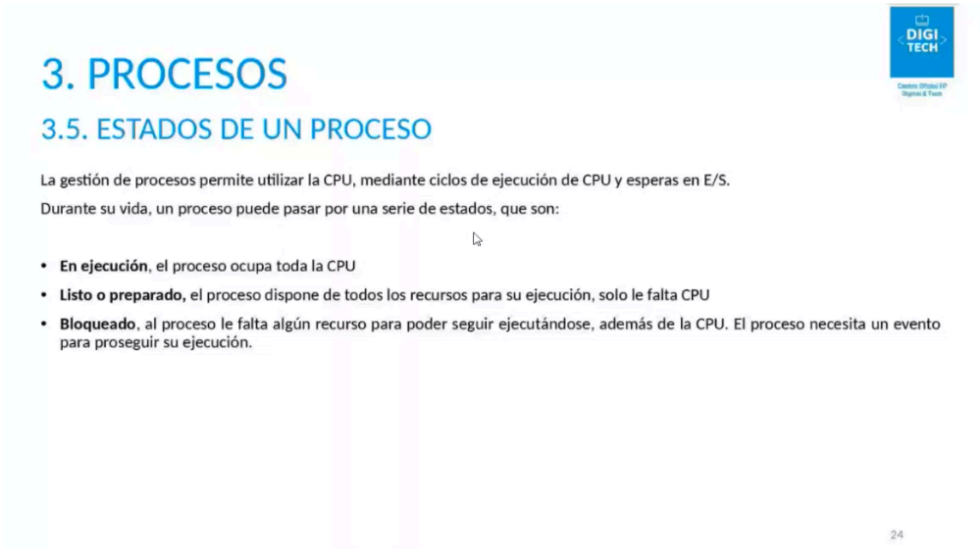
En sistemas modernos, un programa puede crear múltiples procesos. Cada proceso tiene su **propio PCB independiente**, su memoria aislada y sus recursos. Esto ofrece ventajas:

- Mayor seguridad (si un proceso falla, no afecta al resto).
- Mejor aislamiento.
- Posibilidad de ejecutar trabajo paralelo.

El modelo multiproceso, junto al modelo multihilo, es fundamental en modernos sistemas operativos y aplicaciones complejas.

7️⃣ ESTADOS DEL PROCESO

Los estados no solo indican dónde está un proceso, sino que determinan qué decisiones puede tomar el planificador y cómo responde el sistema a los eventos.



Descripción de cada estado

✅ Nuevo

El proceso está en fase de creación. Es un estado inicial y breve, pero técnico. Aquí el sistema operativo prepara todo lo necesario antes de permitir que el proceso entre en la competencia por la CPU. Estas tareas incluyen:

- **Reserva de memoria:** se asignan los segmentos básicos (text, data, bss, heap y stack).
- **Creación del PCB:** se genera la estructura que almacenará toda la información del proceso durante su vida.
- **Carga del ejecutable:** el cargador coloca el código en la zona *text* y deja listas las variables inicializadas.
- **Verificación de permisos:** se valida que el usuario o servicio tenga derecho a ejecutar el programa solicitado.

En esta fase el proceso todavía no puede ejecutarse porque necesita que toda su “infraestructura” esté preparada.

✅ Preparado

En este estado el proceso ya tiene **todo lo necesario** para ejecutarse, excepto una cosa: acceso a la CPU.

- Se encuentra en la **cola de preparados**, esperando turno.
- La mayoría de procesos del sistema pasan gran parte de su vida aquí.
- La duración de la espera depende del **algoritmo de planificación** (Round Robin, prioridades, etc.).

Cuando el planificador lo elige, pasa al estado de ejecución.

✅ Ejecutando

El proceso está **activamente usando la CPU**. Solo un proceso por núcleo puede estar en este estado.

- Ejecuta instrucciones en tiempo real.
- Puede ser **interrumpido** si expira su quantum o el planificador decide priorizar otro proceso.
- Puede ejecutar millones de instrucciones por segundo mientras permanezca aquí.

Es el estado más dinámico: aquí es donde el proceso realmente “trabaja”.

✅ Bloqueado

El proceso no puede seguir ejecutándose porque necesita algo **externo a la CPU**. Por eso se suspende temporalmente.

Causas comunes:

- Lectura o escritura en disco.
- Espera de datos de red.
- Entrada del usuario (teclado, ratón).
- Finalización de una operación en otro proceso.

En este estado **no consume CPU**, lo que permite que otros procesos utilicen ese recurso. Cuando lo que espera se completa, pasa nuevamente a **preparado**.

✅ Terminado

El proceso ha concluido su tarea.

- El sistema libera su memoria asignada.
- Su PCB es eliminado.
- Todos los recursos abiertos (archivos, sockets, pipes) son cerrados automáticamente.

==Después de este punto, el proceso ya no existe en el sistema.

8️⃣ FLUJO DE ESTADOS DEL PROCESO

El flujo de estados describe cómo se mueve un proceso durante su ciclo de vida. No todos los procesos pasan por todos los estados, pero la mayoría sigue este camino general.

Explicación profunda del flujo

1. Nuevo → Preparado

Cuando un proceso es creado, no puede ejecutarse inmediatamente. Primero el sistema operativo debe asignarle los recursos básicos:

- memoria para sus segmentos,
- un PCB para almacenar su estado,
- inicializar su contexto de ejecución.

Cuando todo esto está listo, el proceso pasa a **Preparado**, donde ya está listo para ejecutar pero aún no tiene acceso a la CPU. Es como estar en la “sala de espera”.

---

2. Preparado → Ejecutando

Aquí entra en acción el **planificador**. Este módulo del sistema operativo analiza la cola de procesos listos y selecciona uno para ocupar la CPU.

Las políticas de planificación más comunes:

- **Round Robin:** cada proceso recibe un quantum de tiempo fijo.
- **Prioridades:** procesos con mayor prioridad pasan antes.
- **FIFO:** el primero que llega es el primero que se ejecuta.
- **SJF (Shortest Job First):** favorece a procesos cortos para optimizar tiempos.

El proceso elegido pasa a **Ejecutando** y la CPU empieza a procesar sus instrucciones.

---

3. Ejecutando → Terminado

Si el proceso completa todas sus operaciones sin interrupciones externas, llega a su estado final:

- el sistema libera su memoria,
- cierra sus archivos,
- elimina su PCB.

Ya no tendrá más actividad. Es el cierre limpio y normal del ciclo de vida.

---

4. Ejecutando → Bloqueado

Si durante la ejecución el proceso necesita algo que no depende de la CPU—por ejemplo leer de disco, esperar datos de la red o una entrada del usuario—entonces deja de ser eficiente mantenerlo ejecutándose.

El SO lo mueve a **Bloqueado** mientras espera que la operación externa termine. Así, la CPU queda libre para otros procesos.

---

5. Ejecutando → Preparado

Este cambio ocurre cuando el proceso **pierde la CPU pero no porque esté esperando E/S**, sino porque la CPU debe reasignarse.

Motivos típicos:

- **Se agota su quantum:** en Round Robin el tiempo asignado se agota.
- **El planificador reasigna:** otro proceso tiene mayor prioridad.
- **Interrupciones:** hardware o software interrumpen la ejecución.

El proceso vuelve a la cola de **Preparado**, esperando su siguiente turno.

---

6. Bloqueado → Preparado

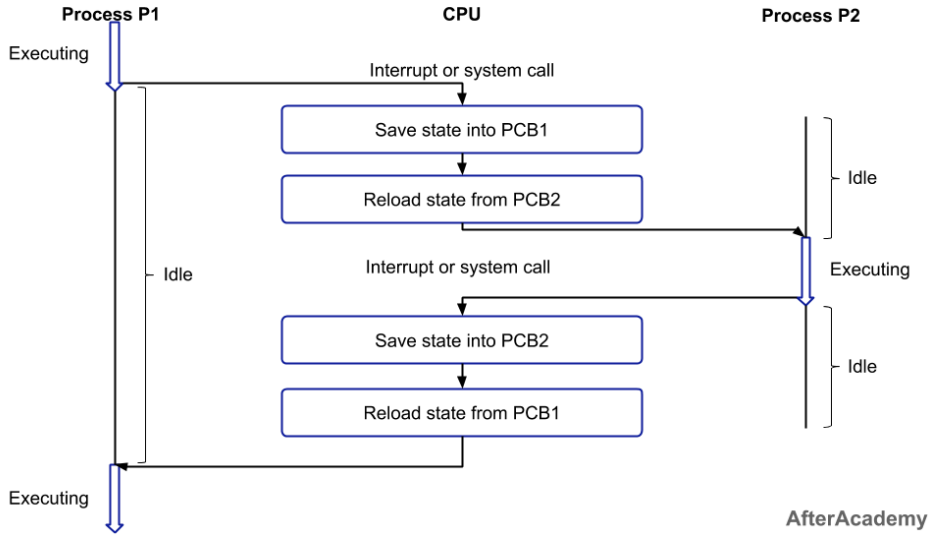
Cuando termina la operación que estaba esperando (por ejemplo, termina la lectura de un archivo), el proceso queda listo para continuar.

Vuelve a **Preparado**, y desde ahí esperará a que el planificador lo seleccione nuevamente para ejecutar.

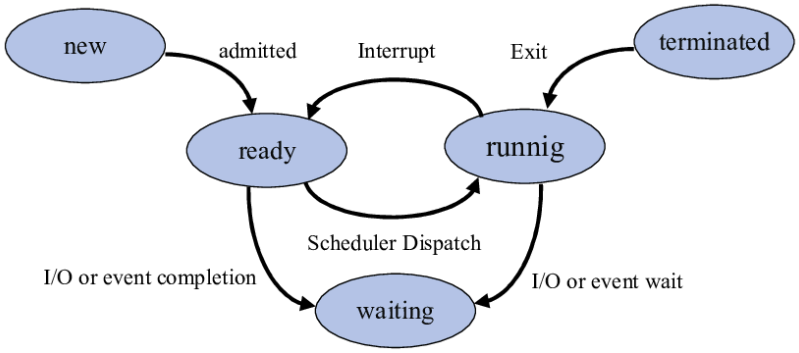
Reglas obligatorias del flujo

- Un proceso **bloqueado nunca puede regresar directamente a ejecutando**.

- Solo los procesos en **preparado** pueden ser elegidos por la CPU.
- Las transiciones están controladas únicamente por el planificador y el SO.



Importancia del modelo de estados



Este modelo garantiza:

- Uso eficiente de la CPU.
- Respuesta rápida a eventos del sistema.
- Capacidad de ejecutar miles de procesos simultáneamente.

Normas importantes

- Un proceso **bloqueado nunca puede ir directamente a ejecución**.
- Si se interrumpe, vuelve a preparado.
- Si finaliza alguna operación externa, regresa a la cola de preparados.

Este modelo permite que el sistema pueda gestionar decenas o cientos de procesos sin perder control sobre ninguno.

9️⃣ CAMBIO DE CONTEXTO

El **cambio de contexto (context switch)** es una operación interna del sistema operativo donde se detiene la ejecución de un proceso y se transfiere el control de la CPU a otro proceso. Este procedimiento no forma parte del trabajo del usuario ni del programa; es un mecanismo del kernel que garantiza la multitarea. El objetivo es que varios procesos compartan la CPU de forma ordenada y segura.

Un cambio de contexto implica que la CPU abandona un proceso y empieza a ejecutar otro, lo que obliga al sistema a guardar toda la información del proceso saliente y restaurar la del proceso entrante. Esto requiere precisión absoluta, ya que cualquier error provocaría comportamientos imprevisibles o corrupción del sistema.

¿Qué ocurre exactamente en un cambio de contexto?

El procedimiento se puede desglosar en etapas claras y sucesivas:

1. **Guardar el estado del proceso actual en su PCB:**
   - Se guardan los registros de CPU, el contador de programa, el estado de la memoria y los recursos abiertos.
   - Queda listo para continuar exactamente desde el punto donde lo dejó.
2. **El scheduler selecciona un nuevo proceso:**
   - Consulta políticas de planificación (prioridades, quantum, colas).
   - Escoge un proceso que está en estado preparado.
3. **Restaurar el contexto del proceso seleccionado:**
   - El sistema carga sus registros, su contador de programa y su estado de memoria.
   - El proceso recupera su entorno exacto.
4. **Reanudar la ejecución:**
   - El proceso continúa su ejecución como si nunca hubiese sido detenido.

Estas operaciones deben realizarse en microsegundos para que el usuario perciba un sistema fluido.

Causas típicas de un cambio de contexto

Un cambio de contexto no se ejecuta por capricho. Es una decisión controlada del sistema operativo que responde a eventos específicos que hacen necesario detener un proceso y ceder la CPU a otro.

✅ Expira el quantum

En planificadores como **Round Robin**, cada proceso recibe un intervalo de tiempo fijo (quantum) para ejecutar. Cuando se consume ese tiempo:

- El SO interrumpe el proceso sin importar si ha terminado su tarea.
- Guarda su estado en el PCB.
- Devuelve el proceso a la cola de preparados.
- Selecciona otro proceso para usar la CPU.

Esto garantiza que ningún proceso monopolice el procesador.

✅ Llega un proceso de mayor prioridad

Cuando aparece un proceso con prioridad superior:

- El SO **interrumpe inmediatamente** al proceso actual.
- La CPU se reasigna sin esperar a que termine la instrucción o a que finalice el quantum.

Ejemplo: un proceso del sistema operativo o una interrupción crítica.

Esto mantiene una correcta respuesta del sistema ante tareas urgentes.

✅ El proceso en ejecución debe esperar

Si un proceso empieza una operación de **E/S, teclado, red o espera un evento de otro proceso**, no puede continuar usando la CPU. El SO:

- Detecta que la CPU ya no aporta nada útil.
- Mueve el proceso a bloqueado.
- Asigna la CPU a otro proceso listo.

De este modo se evita desperdiciar ciclos de CPU.

En todos los casos, el objetivo es equilibrar rendimiento, equidad y rapidez de respuesta para que el sistema se mantenga estable y eficiente.

Coste del cambio de contexto

Un cambio de contexto implica operaciones reales: guardar registros, cargar registros, actualizar estructuras del kernel y conmutar tablas de memoria. Todo esto consume ciclos de CPU.

- En Windows, cada cambio de contexto dura aproximadamente **5 µs**.
- Puede parecer poco, pero en un sistema con cientos o miles de procesos, los cambios pueden alcanzar **decenas de miles por segundo**.

Cuando esto ocurre, la CPU invierte más tiempo en tareas internas del sistema que en ejecutar aplicaciones del usuario.

Por eso, **un exceso de cambios de contexto reduce el rendimiento global**, ya que la CPU queda ocupada en gestionar cambios en lugar de avanzar en cálculos o tareas útiles.

🔟 TRANSICIONES DE ESTADO

Las transiciones de estado describen los cambios que sufre un proceso a medida que avanza su ciclo de vida. Cada transición ocurre como reacción a un evento específico generado por el propio proceso, por el sistema operativo o por una operación de entrada/salida.

👉 De ejecución a bloqueado

Esta transición ocurre cuando el proceso en ejecución se encuentra con una operación que no puede resolver usando la CPU. Debe esperar por un recurso externo. Ejemplos:

- Solicita leer un archivo desde el disco.
- Espera datos de la red.
- Espera entrada del usuario (teclado, ratón).

En esta situación, continuar ejecutándolo sería inútil, porque la CPU no puede avanzar mientras la operación externa no termine. Por ello, el SO lo mueve al estado **bloqueado**, liberando la CPU para otros procesos.

👉 De ejecución a listo

Este cambio sucede cuando el proceso debe liberar la CPU aun siendo capaz de seguir ejecutándose. Las causas más comunes son:

- **Ha agotado su quantum:** el planificador interrumpe su ejecución.
- **Otro proceso de mayor prioridad aparece:** el SO debe ceder la CPU a ese proceso.

En este estado, el proceso regresa a la cola de preparados, manteniendo todo su contexto guardado para continuar más adelante.

👉 De listo a ejecución

El planificador selecciona un proceso de la cola de preparados para que ocupe la CPU. Esta decisión depende del algoritmo de planificación vigente, como Round Robin, FIFO o prioridades.

Una vez seleccionado, el sistema restaura su contexto y el proceso entra en el estado **ejecución**, reanudando su trabajo.

👉 De bloqueado a listo

Un proceso bloqueado se mantiene detenido hasta que se completa la operación por la cual estaba esperando. Una vez ocurre el evento esperado:

- Finaliza la lectura del disco.
- Se reciben los datos de la red.
- La operación de E/S termina.

Entonces el proceso se mueve a **listo**, ya que ahora tiene todo lo necesario para continuar y puede volver a competir por la CPU.

1️⃣1️⃣ TIEMPO DE RÁFAGA

El **tiempo de ráfaga** describe cómo un proceso utiliza la CPU en periodos discretos de actividad. Un proceso no consume CPU de forma continua; en realidad trabaja a saltos: ejecuta unas instrucciones, luego espera, luego vuelve a ejecutar, y así sucesivamente. Este comportamiento es fundamental para entender cómo planifica el sistema operativo.

Definición formal

Una **ráfaga de CPU** es el intervalo de tiempo durante el cual un proceso está usando activamente la CPU para ejecutar instrucciones. Aquí ocurren operaciones lógicas, aritméticas, control de flujo y manipulación de datos en registros.

Entre estas ráfagas de CPU, el proceso cambia inevitablemente a fases donde no puede avanzar sin recursos externos. Este periodo se denomina **ráfaga de E/S**. En una ráfaga de E/S el proceso no usa la CPU y pasa al estado **bloqueado**.

Tiempo de espera (E)

El tiempo de espera es el tiempo total que el proceso pasa en la cola de preparados, esperando a que el planificador le asigne la CPU. Este tiempo afecta directamente al rendimiento y es utilizado por ciertos algoritmos (como SJF o prioridades dinámicas) para reajustar prioridades y minimizar la espera acumulada.

Dinámica del ciclo de ejecución

La ejecución real de un proceso se compone de una secuencia alternada:

- 🟦 **Ráfaga de CPU:** el proceso progresa ejecutando instrucciones. En este estado está consumiendo recursos del procesador. Es donde se calculan operaciones, se actualizan estructuras de datos y se ejecutan algoritmos.
- 🟨 **Ráfagas de E/S:** el proceso queda bloqueado esperando la respuesta de un dispositivo externo (disco, red, teclado, etc.). Durante este tiempo no progresa y libera la CPU.

Las duraciones de las ráfagas varían mucho de un proceso a otro, pero suelen seguir patrones estadísticos predecibles que los planificadores utilizan para optimizar el rendimiento general del sistema.

1️⃣2️⃣ PLANIFICACIÓN DE PROCESOS

La planificación de procesos es el mecanismo mediante el cual el sistema operativo decide **qué proceso usa la CPU y cuándo**. Dado que la CPU es un recurso finito y los procesos pueden ser miles, el planificador debe establecer un orden racional para garantizar fluidez, eficiencia y equidad.

El componente que ejecuta esta tarea es el **planificador** (scheduler) y el **dispatcher** (que realiza el cambio de proceso). Ambos colaboran: el scheduler toma la decisión; el dispatcher la ejecuta.

¿Por qué es tan importante la planificación?

El rendimiento global del sistema depende directamente de cómo se asigna la CPU. Una mala planificación puede provocar:

- tiempos de espera largos
- baja capacidad de respuesta
- sobrecarga de cambios de contexto
- sensación de lentitud en el usuario

El objetivo del planificador es equilibrar:

- rapidez de respuesta
- eficiencia global
- prioridad de procesos críticos
- justicia entre procesos

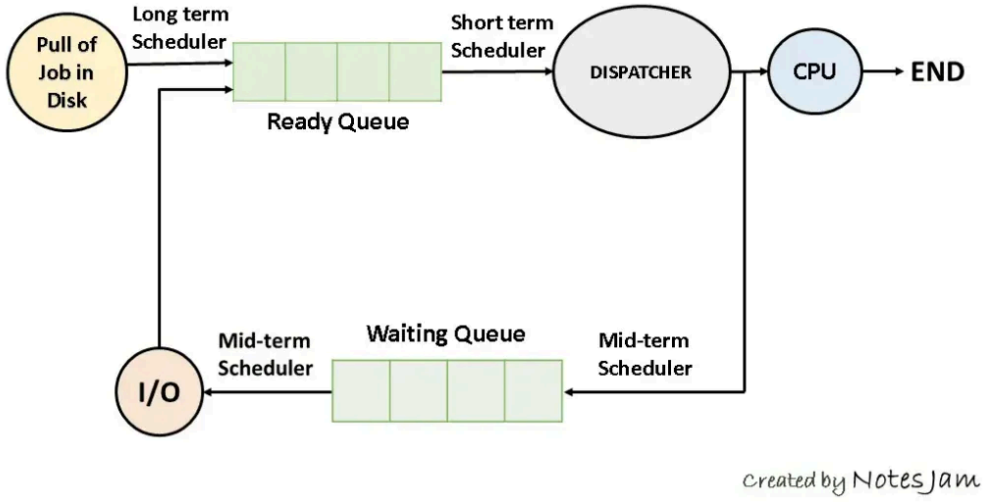
¿Qué provoca la llamada al dispatcher?

El dispatcher solo se activa cuando debe producirse un cambio de proceso. Esto ocurre en cuatro situaciones específicas:

- **El proceso termina su ejecución:** ya no requiere CPU y debe retirarse.
- **El proceso se bloquea:** necesita E/S o un recurso externo y ya no puede avanzar.
- **Agota su quantum:** en políticas como Round Robin, debe ceder la CPU.
- **Otro proceso se activa en estado listo:** puede ser por prioridad superior o por finalización de E/S.

En todos estos casos, el dispatcher ejecuta un **cambio de contexto**, que supone guardar el estado del proceso actual y restaurar el del siguiente.

Esta imagen muestra **cómo fluye un proceso** por el sistema operativo y **cómo actúan los tres tipos de planificadores**:



✅ 1. Long-term scheduler (planificador a largo plazo)

- Se encarga de **admitir nuevos trabajos** desde el disco hacia la **cola de listos (Ready Queue)**.
- Controla cuántos procesos entran al sistema y evita sobrecargar la RAM.

✅ 2. Short-term scheduler (planificador a corto plazo)

- Selecciona un proceso de la **cola de listos**.
- Se lo entrega al **dispatcher**.
- El dispatcher lo pone a ejecutar en la **CPU**.

Es el que toma decisiones con más frecuencia.

✅ 3. Mid-term scheduler (planificador a medio plazo)

- Se encarga de procesos que están en la **cola de espera (Waiting Queue)**.
- Puede **suspender** procesos (swapping) o reactivarlos.
- Gestiona procesos bloqueados por **E/S (I/O)**.

Optimiza la RAM moviendo procesos entre memoria y disco.

> ACLARACIÓN:

I/O significa **Input/Output**, es decir, **Entrada/Salida**.

En un sistema operativo representa todas las operaciones en las que un proceso **necesita interactuar con un dispositivo externo**, como por ejemplo:

Entrada (Input):

- Teclado
- Ratón
- Sensor
- Archivo leído desde el disco
- Datos recibidos por red

Salida (Output):

- Escritura en disco
- Enviar datos por red

- Mostrar algo en pantalla
- Imprimir

Cuando un proceso realiza I/O:

- **No puede continuar usando la CPU**, porque está esperando la respuesta del dispositivo.
- Pasa al estado **bloqueado** hasta que la operación se complete.

Por eso en los esquemas aparece la burbuja “I/O”: es el punto donde el proceso está esperando operaciones de entrada/salida.

🔄 Flujo general

1. Los trabajos llegan desde disco → pasan al **long-term scheduler**.
2. Entran en la **cola de listos**.
3. El **short-term scheduler** elige uno.
4. El **dispatcher** lo envía a ejecutar en la **CPU**.
5. Si necesita E/S, va a la **waiting queue** (bloqueado).
6. El **mid-term scheduler** gestiona estos bloqueos.
7. Cuando termina la E/S vuelve a la **cola de listos**.
8. Cuando termina su ejecución → **END**.

1️⃣3️⃣ USO EFICIENTE DEL DISPATCHER

El profesor plantea una cuestión central para el rendimiento del sistema:

> ¿Es conveniente llamar constantemente al dispatcher?

La respuesta es clara desde el punto de vista de la eficiencia.

Respuesta explicada

Cada llamada al dispatcher implica un cambio de contexto. Este proceso consume tiempo de CPU y no aporta valor directo al usuario. Por tanto:

- **Más llamadas al dispatcher = más tiempo perdido en tareas internas.**
- **Menos llamadas = más tiempo de CPU dedicado a trabajo real.**

En otras palabras, se debe evitar el uso excesivo del dispatcher porque obliga a la CPU a dedicar ciclos a gestionar el sistema en lugar de ejecutar procesos productivos.

Buen uso del dispatcher

- Permitir que los procesos ejecuten todo lo posible sin interrupciones artificiales.
- Evitar quantum demasiado cortos que provoquen recambios constantes.
- Reducir procesos innecesarios o de prioridad baja.
- Favorecer escenarios donde un proceso pueda: → **Ejecutar y terminar** sin ser interrumpido.

Con esto se obtiene un sistema operativo más fluido y eficiente, donde la CPU se emplea principalmente en tareas del usuario y no en labores de gestión interna.
